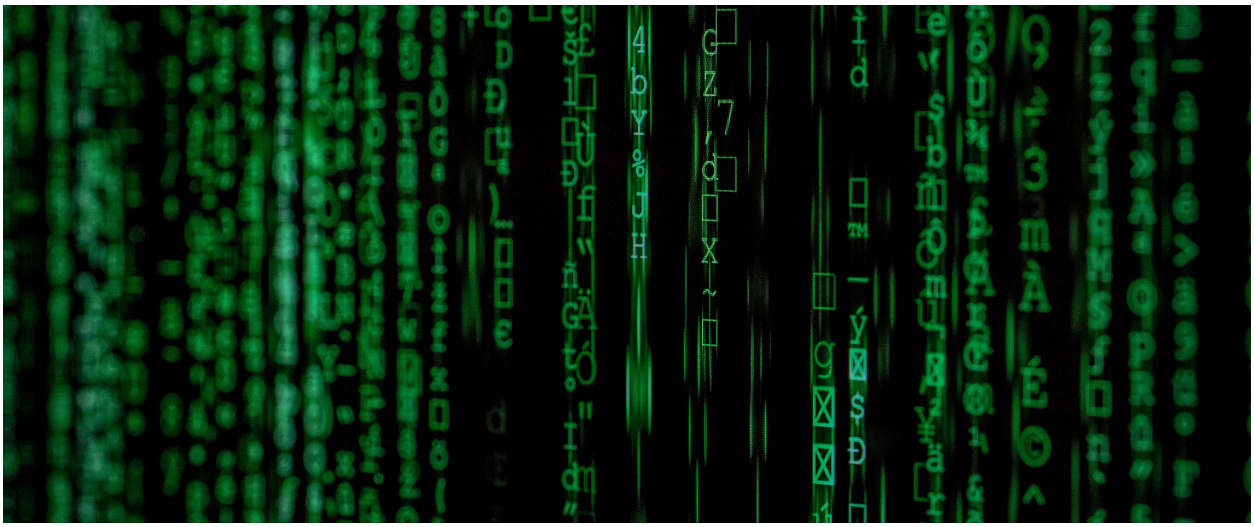

Reinforcement Learning and Q learning- A blog

Introduction

Reinforcement learning is a subfield of machine learning that deals with learning through interactions with an environment. The goal is to learn a policy that maps states to actions in order to maximize a cumulative reward signal. It is a fascinating area of machine learning that has attracted a lot of attention in recent years. It is a type of learning where an agent learns how to make decisions in an environment by receiving feedback in the form of rewards or punishments. Reinforcement learning has a wide range of applications, from game playing to robotics to finance.

Q learning is a popular reinforcement learning algorithm that uses a table to store the expected reward for each state-action pair. The algorithm updates the table using the Bellman equation and uses a greedy policy to select actions. Q learning has been successfully applied to a wide range of problems, including game playing, robotics, and control.

Our aim to is to design a Q learning model which is able to classify roads as clean and dirty. For this, we have been assigned a dataset (https://www.kaggle.com/datasets/faizalkarim/cleandirty-road-classification) which consists of 237 images out of which 113 were labelled clean and 124 were labelled dirty.

Classifying roads as clean or dirty can have several benefits. Firstly, it helps in identifying the areas that need immediate attention and cleaning. If a road is classified as dirty, then the concerned authorities can take necessary actions to clean it up and maintain hygiene. Secondly, it can help in preventing the spread of diseases caused by unhygienic conditions. A dirty road can attract pests, bacteria and viruses that can cause various illnesses. By classifying roads as clean or dirty, preventive measures can be taken to ensure that the environment is safe and healthy for people to live in. Lastly, clean roads contribute to the aesthetic beauty of a place and help create a positive image of the community, which can attract tourists and investors.

Problem Statement

We need to implement a Q Learning algorithm to classify the cleanliness of a road in a given image. We hence, need to develop an autonomous agent using Q-learning algorithm, which is able to learn from a dataset of road images and output whether a new image represents a clean or a dirty road.

About the Dataset

The images were collected from various sources and were taken under different lighting conditions, weather conditions, and from different camera angles. The images are in JPEG format and have a resolution of 224x224 pixels. The images have been pre-processed and resized to this resolution, making them suitable for use with deep learning algorithms.

The dataset also includes a CSV file that contains the file names and corresponding labels for each image. This file can be used to train and test machine learning models.

One of the strengths of this dataset is its diversity. The images were collected from various sources, and the road conditions vary widely. Some images were taken on rural roads, while others were taken on busy city streets. This diversity ensures that the resulting models are robust and can generalize well to new, unseen data.

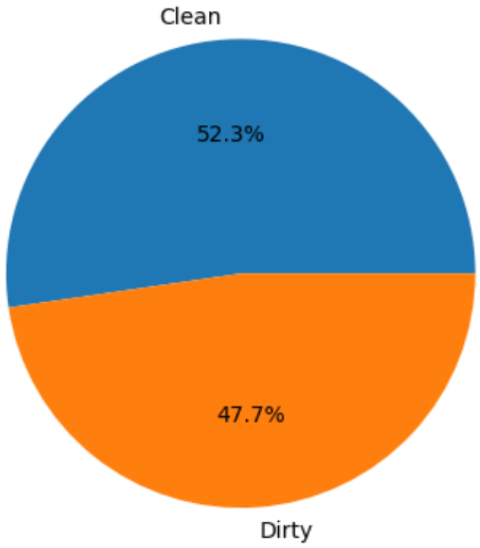

fig.1) percentage of clean and dirty roads

The above graph shows that the dataset provided is almost balanced, hence no further processing was done in order to make it balanced.

Implementation of Q Learning

Q-learning is a popular reinforcement learning algorithm that uses a table to store the expected reward for each state-action pair. The algorithm updates the table using the Bellman equation and uses a greedy policy to select actions. In Q-learning, the agent learns an action-value function that gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. Q-learning is a model-free reinforcement learning algorithm, which means that it does not require a model of the environment. Instead, it learns by interacting with the environment and observing the rewards that it receives. Q-learning is particularly well-suited for problems where the state and action spaces are small enough to be represented in a table, making it a popular choice for many reinforcement learning applications.

To implement q learning on the given dataset, we do consider the condition of the road as the state and the classification of the road as clean and dirt, as the action. We can the find the value of Q for a particular state and its action using-

$$Q(s,a) = R(s,a) + \gamma * max(Q(s',a')$$

Where in-
Q (s, a)- value of Q for the state and its corresponding action
R (s, a)-reward for the state and its corresponding action
γ- discount factor
max (Q(s' ,a '))- maximum value of Q for next states and all possible action.
γ is the discount factor which determines the weights to be applied to the future rewards.

For this dataset, we have applied reward of +1 for correct classification for clean road and reward of -1 for incorrect classification for clean road. Likewise, we have applied reward of +1 for correct classification for dirty road and reward of -1 for incorrect classification for dirty road.

Implementation in Python

a) Importing the required libraries

To implement Q learning on our dataset, we have used python. In order to do this, we first needed to implement the required libraries.

```
import gym
import numpy as np
import cv2
import os
```

We have used "gym" which Python module that provides a collection of environments for developing and comparing reinforcement learning algorithms. It offers a simple interface for defining custom environments, and also includes a range of pre-built environments for tasks such as playing Atari games or controlling robotic arms. The module also provides tools for visualizing and evaluating the performance of reinforcement learning agents.

b) Creating the Environment

For our dataset, there was no predefined environment present. so we created our own Environment called “RoadEnvironment” that simulates a road that can be clean or dirty. The environment has two possible states, 0 for clean and 1 for dirty, and two possible actions, 0 for clean and 1 for dirty.

```
# Define the environment
class RoadEnvironment(gym.Env):
    def __init__(self, img_dir):
        self.observation_space = gym.spaces.Discrete(2) # 0 = clean, 1 = dirty
        self.action_space = gym.spaces.Discrete(2) # 0 = clean, 1 = dirty
        self.state = None
        self.reward_range = (-1, 1)
        self.current_step = 0
        self.img_dir = img_dir
        self.img_paths = sorted([os.path.join(img_dir, f) for f in os.listdir(img_dir)])
        self.max_steps = len(self.img_paths)

    def reset(self):
        self.current_step = 0
        self.state = 0 if "clean" in self.img_paths[self.current_step] else 1
        return self.state

    def step(self, action):
        if action == 0:
            reward = 1 if self.state == 0 else -1
        else:
            reward = 1 if self.state == 1 else -1

        self.current_step += 1
        done = self.current_step >= self.max_steps
        if not done:
            self.state = 0 if "clean" in self.img_paths[self.current_step] else 1
        next_state = self.state
        return next_state, reward, done, {}
```

The `__init__` method of the environment initializes the environment by defining the observation space, action space, reward range, and the maximum number of steps. In this environment, the observation space and action space are both discrete, with two possible values: 0 for clean and 1 for dirty. The reward range is set to -1 to 1, meaning that the rewards are either negative, zero or positive. The maximum number of steps is set to the total number of images in the image directory.

The `reset` method is called at the beginning of each episode and returns the initial state of the environment. In reinforcement learning, an episode is a sequence of steps starting from the initial state and ending in a terminal state. It is a unit of interaction

between an agent and an environment. In this case, the initial state is set to 0 if the first image in the image directory is clean, otherwise it is set to 1.

The `step` method is called at each time step and takes an action as input. The method updates the state of the environment based on the action taken by the agent, and returns the next state, reward, done flag, and an empty dictionary of additional information. If the action taken is 0, the reward is 1 if the current state is 0 (clean), and -1 otherwise. If the action taken is 1, the reward is 1 if the current state is 1 (dirty), and -1 otherwise. The done flag is set to True when the maximum number of steps has been reached. The next state is set to the current state if the maximum number of steps has been reached, otherwise it is set to 0 if the next image in the directory is clean, and 1 otherwise.

c) Defining the Agent

Next, we define a Q learning agent called “QLearningAgent” that learns to take the appropriate action given a certain state. The agent initializes a Q-table with zeros and uses the Q-learning algorithm to update the table as it interacts with the environment.

```
# Define the Q-learning agent
class QLearningAgent:
    def __init__(self, alpha=0.1, gamma=0.99, epsilon=1.0):
        self.q_table = np.zeros((2, 2))
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon

    def act(self, state):
        if np.random.uniform() < self.epsilon:
            action = np.random.choice(self.q_table.shape[1])
        else:
            q_values = self.q_table[state, :]
            action = np.argmax(q_values)
        return action

    def learn(self, state, action, reward, next_state, done):
        q_next_max = np.max(self.q_table[next_state, :])
        q_current = self.q_table[state, action]
        q_new = q_current + self.alpha * (reward + self.gamma * q_next_max - q_current)
        self.q_table[state, action] = q_new

        if done:
            self.epsilon *= 0.99
```

The agent's constructor takes in three hyperparameters: alpha, gamma, and epsilon, which determine the learning rate, discount factor, and exploration rate, respectively.

In machine learning, Learning Rate is a hyperparameter which determines how quickly or slowly the model learns from the data. Discount rate and Exploration rate are also hyperparameters, mainly used in Reinforcement Learning, wherein, the exploration rate determines the probability that the agent will choose a random action rather than the action that has the highest expected reward based on its current knowledge. on the other hand, Discount rate determines the importance of future rewards in the agent's decision-making process.

The agent has three methods: act(), learn(), and an initialization method (**init**()). The act() method takes in the current state and selects an action based on the epsilon-greedy policy. The learn() method updates the Q-value table based on the Bellman equation using the state, action, reward, and next state obtained from the environment.

The Q-value table is represented as a 2D NumPy array, with rows corresponding to states and columns corresponding to actions. The **init**() method initializes the Q-value table to zeros.

d) Creating the Instances

An instance of the `RoadEnvironment` class is created with the path to the image directory as the argument. This environment is then used to initialize an instance of the `QLearningAgent` class with default values for the learning rate (`alpha`), discount rate (`gamma`), and exploration rate (`epsilon`).

```
# Initialize the environment and agent
img_dir = "/content/drive/MyDrive/Deep Learning/Dataset/images"
env = RoadEnvironment(img_dir)
agent = QLearningAgent()
```

e) Training the Agent

In order to make the agent learn to predict whether a road is clean or dirty, we need to train it. The agent is trained on the environment for a set number of episodes, and then tested on a new image.

```
# Train the agent
episodes = 1000
for episode in range(episodes):
    state = env.reset()
    total_reward = 0
    done = False

    while not done:
        action = agent.act(state)
        next_state, reward, done, _ = env.step(action)
        agent.learn(state, action, reward, next_state, done)
        state = next_state
        total_reward += reward

    print(f"Episode: {episode + 1}/{episodes}, Total Reward: {total_reward}")
```

The Q Learning agent is run for a specified number of episodes, initializes the state of the environment using `env.reset()`, and starts the agent's action selection process by calling `agent.act(state)`. After the agent selects an action, the environment is updated using `env.step(action)`, and the agent learns from the updated state of the environment using `agent.learn(state, action, reward, next_state, done)`.

The total reward obtained in each episode is stored in the `total_reward` variable, which is printed at the end of each episode using `print(f"Episode: {episode + 1}/{episodes}, Total Reward: {total_reward}")`.

```
Episode: 1/1000, Total Reward: 1
Episode: 2/1000, Total Reward: -1
Episode: 3/1000, Total Reward: -1
Episode: 4/1000, Total Reward: 1
Episode: 5/1000, Total Reward: 1
Episode: 6/1000, Total Reward: -1
Episode: 7/1000, Total Reward: 1
Episode: 8/1000, Total Reward: -1
Episode: 9/1000, Total Reward: 1
Episode: 10/1000, Total Reward: 1
Episode: 11/1000, Total Reward: 1
Episode: 12/1000, Total Reward: 1
Episode: 13/1000, Total Reward: 1
Episode: 14/1000, Total Reward: 1
Episode: 15/1000, Total Reward: 1
```

fig.2) snippet of the output for the above code block

The above figure displays the episode number and the total reward obtained by the agent during that episode. The total reward is the sum of all rewards obtained by the agent while interacting with the environment.

f) Testing the Agent

Finally, we test our newly defined agent, on a new image. The new image is loaded into memory, and the Q-learning agent predicts the appropriate action to take given the state of the road in the image.

```
# Test the agent on a new image
new_img_path = "/content/drive/MyDrive/Deep Learning/Dataset/images/test_clean.jpg"
new_state = 0 if "clean" in new_img_path else 1
action = agent.act(new_state)
if action == 0:
  prediction = "Clean"
else:
  prediction = "Dirty"
print(f"The Road is: {prediction}")
```

After the new image is loaded, the code the passes the corresponding state (0 for clean, 1 for dirty) to the agent's act method to get an action. If the action is 0, the code predicts that the road is clean; if the action is 1, it predicts that the road is dirty. Finally, it prints the prediction.

```
The Road is: Clean
```

fig 3.) the predicted output

Thus, the output for the new image says that the roads in the image are clean

Conclusion

To conclude, we have successfully implemented a Q-learning algorithm on a dataset of road images to classify the cleanliness of the road. We have used the Q-learning algorithm to learn a policy that maps image features to actions of classifying the road as clean or dirty. We have also discussed the implementation of the Q-learning algorithm in Python, along with the necessary libraries and steps involved in creating an environment. The Q-learning algorithm is a powerful tool in the field of reinforcement

learning, and its ability to learn from interactions with the environment makes it suitable for a wide range of applications. In the context of road classification, the Q-learning algorithm provides an efficient and accurate means of classifying roads as clean or dirty, which has important implications for public health and safety.